ARPA ORDER NO. 189-1

MEMORANDUM
RM-6104-ARPA
JANUARY 1970

USER'S MANUAL FOR APAREL: A PARSE-REQUEST LANGUAGE

R. M. Balzer

PREPARED FOR:
ADVANCED RESEARCH PROJECTS AGENCY

The RAND Corporation
SANTA MONICA • CALIFORNIA

44

ARPA ORDER NO. 189-1

MEMORANDUM
RM-6104-ARPA
JANUARY 1970

USER'S MANUAL FOR APAREL: A PARSE-REQUEST LANGUAGE

R. M. Balzer

This research is supported by the Advanced Research Projects Agency under Contract No. DAHC15 67 C 0111. Views or conclusions contained in this study should not be interpreted as representing the official opinion or policy of ARPA.

The RAND Corporation
1700 MAIN ST. • SANTA MONICA • CALIFORNIA • 90406

PREFACE

This Memorandum describes the use of APAREL, a parsing capability embedded within the PL/I language. The APAREL extension allows users to specify both the syntax of their parse-requests in a BNF-like language and the semantics associated with a successful parse-request in the PL/I language.

The Memorandum is based on the assumption that the reader has read *APAREL--A Parse-Request Language*[†] and that he understands the basic ideas of top-down parsing.

APAREL has been developed as a basic tool for use in man-machine communication studies at The RAND Corporation under the sponsorship of the Advanced Research Projects Agency.

[†]R. M. Balzer, and D. J. Farber, *APAREL--A Parse-Request Language*, The RAND Corporation, RM-5611-1-ARPA, September 1969.

SUMMARY

This Memorandum is a user's manual for APAREL, which is a parse-request language. It describes the features that have and have not been implemented, the restrictions on the use of these facilities, the new features added to APAREL since the publication of *APAREL--A Parse-Request Language*,† the method of invoking the available facilities, and ideas on the effective and efficient use of APAREL.

---

†R. M. Balzer, and D. J. Farber, *APAREL--A Parse-Request Language*, The RAND Corporation, RM-5611-1-ARPA, September 1969.

CONTENTS

I. INTRODUCTION

APAREL is presently implemented as a set of subroutines callable from PL/I. Therefore, APAREL programs must be set up using these calls rather than those specified in *APAREL--A Parse-Request Language* [1]. In addition, certain features mentioned in that publication have not yet been implemented, while certain new features have been added. Also, several implementation restrictions exist. All of the above are detailed in this manual.

The Memorandum is based on the assumption that the reader has read *APAREL--A Parse-Request Language* [1], and that he understands the basic ideas of top-down parsing.

II. USE OF APAREL

All parsing capabilities of APAREL are invoked by calls to the APAREL parser. These calls are used in the following ways: 1) to define, redefine, and delete parse-requests, 2) to define parse-related names, 3) to initiate a parse-request, 4) to terminate the semantics of a parse-request, and 5) to turn the trace of the parsing on or off.

Each call can be given at any time, with the exception of terminating the semantics of a parse-request, which can only be issued from a semantic routine initiated by APAREL as the result of a successful parse-request. Hence, except as noted above, all APAREL functions can be dynamically invoked, providing such features as:

1) Dynamic addition of new parse-requests;
2) Dynamic redefinition of parse-requests;
3) Dynamic tracing of parse;
4) Recursive initiation of parse-requests.

However, since no incremental compiler is available for PL/I, semantic routines cannot be dynamically added, redefined, or deleted. The routines are:

1) DEFINE_PARSE_REQUEST

This routine is used to define or redefine a parse-request or a parse-related name (such as a semantic-routine name). If the parse-request or parse-related name specified has already been defined, it is deleted and defined as if it were new.

This routine has three or four arguments. First is the parse-request being defined, which is passed as a character string. Its form is as specified in *APAREL--A Parse-Request Language* [1], except the double colons at each end are not present. The second argument is a character string into which the results of a successful parse of that parse-request will be placed. Third is a binary fixed variable into which

the number of the successful option of that parse-request will be placed. The fourth argument, if present, is a label in the PL/I program to which control will be passed upon successful completion of the parse-request; i.e., it is the label at the start of the semantic routine for the parse-request. (The routine TERMINATE_PARSE, as explained below, terminates the semantics of a parse-request.) If the first argument consists of only a single name or a single name followed by a colon, it is interpreted as the definition of a parse-related name. Its use in other parse-requests determines its type of parse-related name. These types are:

a) PARSE_NAME: If the name appears followed by a colon, it is interpreted as being a local parse name. The parse results and the parse-results option (as specified in the second and third arguments, respectively) of the call that defined the parse-related name will be set to the parse results of the PARSE_ALTERNATIVE_GROUP in which the PARSE-NAME appeared. A fourth argument, if specified in the defining call, will be initiated as the semantic routine for the parse-related name.

b) PARSE_TIME_ROUTINE_NAME: If the name appears after a semicolon in a parse-request, it is interpreted as being the name of a parse-time semantic routine. The label specified as the fourth argument in the call defining the parse-related name will be initiated as a semantic routine.

c) Indirect parse specification: If neither above condition holds, the parse-related name is treated as the indirect specification of a parse rule, and the current value of the second argument in the defining call of the parse-related name is used as the invoked parse-request.

2) PARSE

This routine, which is used to initiate a parse-request, has three arguments, each of which is a character string. The first is the input string; i.e., the string to be parsed. This string will not be altered by APAREL. The value of the second argument is the parse-request that will be used to parse the input. It can be a complex parse-request or, as is usually the case, simply the name of a previously defined parse-request; it is used merely to invoke that parse-request. The third argument is a character string into which will be placed that portion of the input string that was not parsed successfully. During the parsing of the original parse-request, if any parse-request (the original, any initiated by it, or any they initiate, etc.) is successful and has a semantic routine specified or if a PARSE_TIME_ROUTINE_NAME is encountered, the parse is temporarily suspended, and the semantic routine is initiated. After it returns (see TERMINATE_PARSE below), the parse is resumed.

3) TERMINATE_PARSE

This routine returns control to APAREL from a semantic routine; it has one argument, a binary fixed value. If the value is zero (unsuccessful), APAREL will continue the parse as if the current parse-request had syntactically failed at the current point (further alternatives may still allow the parse-request to be successful). If the value is nonzero (successful), the parse will continue as if the semantics had not been invoked. In either case, if the semantic routine alters the value of the parse results (the second argument in the DEFINE PARSE REQUEST routine), the altered value will be passed to any higher-level parse-requests and used in forming their parse results.

4) DELETE_PARSE_REQUEST

This routine deletes a parse-request; it has one argument--a character string--which is the name of the parse-request to delete.

5) TRACE_PARSE

This routine, used to turn tracing on or off, has no arguments. Each call changes the setting of the trace switch from off to on, or vice versa.

6) COMPILE_PARSE_REQUEST

This routine defines a parse-request just as the define-parse-request routine does; it has the same arguments with the same usage. This routine is used when all alternatives are one character literals and when the parse-request is used frequently. Instead of testing each alternative sequentially until a successful one is found or until all have been tried, this routine builds a translate table [2] to test all alternatives simultaneously in parallel; hence, the speed of the parse is greatly improved.

III. ADDITION AND OMISSION OF FEATURES

The following features have been omitted in the present implementation of APAREL:

1) The BAL function--string balanced with respect to specified arguments.

2) PARSE-REQUEST-SEQUENCES--the user must set up a parse-request that contains, as alternatives, the desired sequence of parse-requests; e.g., if the parse-request sequence A1, A2, A3, A4 is desired, the call

```
PARSE(input, 'A1|A2|A3|A4', remaining_input)
```

will effect the parse-request sequence.

3) INPUT and OUTPUT VARIABLES.

4) The NORMAL SEPARATION and SEMANTICS OPEN or CLOSED statements.

The following features have been added:

1) Ability to redefine parse-requests dynamically through the DEFINE_PARSE_REQUEST routine.

2) Ability to trace a parse-request dynamically.

3) A NOT function--it can be stated in a parse-request that the input must not match a particular PARSE_ELEMENT (specified by the NOT symbol (¬), followed by the PARSE_ELEMENT not wanted). If the PARSE_ELEMENT is successful, the alternative will fail; if the PARSE_ELEMENT is unsuccessful, the parsing of the alternative will continue. For example, in a language with reserved words, and assuming a parse-request called RESERVED_WORD exists to define these words, the definition of an identifier might be

```
identifier:¬reserved_word letter(-ARBNO(alphanumeric,-)|)
```

That is, an identifier is a letter followed optionally by an arbitrary number of alphanumerics separated by NULLs (and with no intervening blanks as specified by the minus signs), which is not a reserved word.

Similarly, to define a relation as an arbitrary number of terms separated by relational operations, but including at least one relational operator (i.e., a single term is not to be a relation), the following parse-request can be used:

```
relation:¬(term¬relational_operator)
                ARBNO(term,relational_operator)
```

4) A termination function--this function, specified by the slash symbol (/) in a parse-request and used to require that the end of the input string be reached, is successful if no nonblank characters remain unparsed in the input string. Furthermore, if the termination function is preceded by a minus sign (-), it will be successful only if the entire input string has been parsed; i.e., no characters remain unparsed.

IV. IMPLEMENTATION RESTRICTIONS

1) A parse-request cannot have more than four nested levels within it. Each nested PARSE_GROUP (a set of alternatives enclosed in '⟨' and '⟩' brackets) or ARBNO function counts as one level. Thus, the parse-request

B: ⟨C|⟨D|E ARBNO(F,G)⟩|H⟩⟨I|J⟩

has three nested levels (two PARSE_GROUPS and the ARBNO function).

2) A PARSE_RESULT's maximum size is 256 bytes.

3) The maximum number of elements in a parse-request, counting one for each PARSE_ATOM, PARSE_NAME, and APAREL syntax operator ('⟨', '⟩', '|', '.', etc.), must not exceed 128.

4) The total number of PARSE_REQUESTS, PARSE_RELATED names, and unique literals within PARSE_REQUESTS must not exceed 2048.

5) PARSE_ALTERNATIVE_NAMES cannot be used.

6) PARSE_TIME routines (specified within a parse-request following a semicolon) must have no parameters.

7) All semantic routines must be in the same PL/I block as the call to PARSE, which initiated the parse-request invoking the semantic routines.

8) The ARB function may not appear inside an ARBNO function immediately. It can be used inside an ARBNO function if it is a PARSE_ATOM, which is part of a PARSE_GROUP (i.e., it is enclosed in a pair of '⟨' '⟩' brackets). Thus,

ARBNO(⟨A|ARB B|C⟩,D)

is acceptable.

9) Normal separation is assumed to be zero. Hence, if one or more blanks are desired, the period notation must be used.

10) A PARSE_NAME cannot be specified for the ARB function. For any other PARSE_ATOM, this can be accomplished by preceding the PARSE_ATOM with a parse name and enclosing the pair in PARSE_GROUP brackets (e.g., (name: atom)). This method of naming (via the right-angle bracket) ends the parse group in which the parse atom occurs and, as explained in Sec. V, prevents the ARB function from working correctly.

V. PROGRAMMING CONSIDERATIONS

To use APAREL effectively, the user should be aware of its basic method of parsing. The two types of backup in parsing are: 1) when the input pointer backs up as mismatches are encountered, and 2) when the PARSE_RULE (or its equivalent) pointer backs up. APAREL uses only the first of these; i.e., the PARSE_RULE pointer moves strictly left to right through a parse rule (two exceptions are explained below). Within a PARSE_ALTERNATIVE_GROUP, each alternative is tried until one is found that is successful (e.g., in the parse-request

```
NAME:(A1|A2|A3)B1|B2
```

A2 is successful). Then the parser skips to the end of the PARSE_ALTERNATIVE list (the bracket after A3) and processes the next PARSE_ELEMENT (B1), if any, in the PARSE_ELEMENT_LIST. If this PARSE_ELEMENT (B1) fails, the parser will again skip to the next alternative (B2) in that PARSE_ALTERNATIVE_LIST. It will not go back and try alternative A3 followed by B1; thus, the ordering of alternatives in a PARSE_REQUEST is important. If one of two alternatives can match a prefix of the input that the other can match, the second alternative should be placed before the first in a PARSE_ALTERNATIVE_LIST; e.g., the alternatives A1 and A1 A2 should be ordered

```
A1 A2|A1
```

in a PARSE_REQUEST. The "longest" or "biggest" alternatives should be placed first.

The ARBNO and ARB functions are the two exceptions to the strict left to right movement of the PARSE_RULE pointers. The ARBNO function matches an arbitrary but nonzero number

of occurrences of the first argument; these occurrences are separated by occurrences of the second argument. The PARSE_REQUEST pointer will alternate between these arguments until one fails (if the first argument fails, the input pointer is backed up past the last occurrence of the second argument). The PARSE_REQUEST pointer will then skip past the right parenthesis after the second argument.

The ARB function, which matches an arbitrary string, matches first a string of zero length, and the parse_request pointer moves to the next PARSE_ELEMENT in the PARSE_ELEMENT list (e.g., in the PARSE_REQUEST

NAME: A1 ARB A2(B1|B2)A3|A4

this would be A2). If this PARSE_ELEMENT or any further one (say A3) in the PARSE_ELEMENT_LIST fails, the PARSE_REQUEST pointer is backed up to the ARB; the length of the string that the ARB matches is increased by one; and the PARSE_REQUEST pointer again moves to the next PARSE_ELEMENT (A2) in the PARSE_ELEMENT_LIST. This process is repeated until either the entire PARSE_ELEMENT_LIST succeeds or until the ARB runs out of input to match, in which case the PARSE_ELEMENT_LIST fails. In either case, processing continues, as with normal PARSE_ELEMENT lists.

Left-recursion is handled uniquely. The state of the parser is determined by two variables: 1) the position in the input string, and 2) the position in the parse-request. Before attempting a match for any alternative, the parser checks to see if the present state has occurred before (during the current initiation of the original parse-request). If it has, a left recursive loop has occurred and the parser simply moves on to the next alternative to break the left recursive loop. Therefore, this would cause the rule

number: number digit|digit

to fail on more than two-digit numbers. This can be remedied by using the ARBNO function, which allows iterative specification rather than nested recursive definition; thus,

```
number:  ARBNO(digit,-)
```

A number is an arbitrary nonzero number of digits separated by NULLs (the minus sign ensures that no embedded blanks are in the number); or, even more elegantly:

```
expression:  ARBNO(expression,operator)|(expression)
             |variable|number
             |unary_operator expression
```

An expression is an arbitrary nonzero number of expressions separated by operators, a parenthesized expression, a variable, a number, or a UNARY_OPERATOR followed by an expression.

Care also must be exercised with semantic routines, those specified as PARSE_TIME semantics, or those specified as semantic routines for PARSE_REQUESTS. After they are invoked and have returned, as the parse continues, the input for which they were invoked may be backed up past. It may then be reparsed or it may remain as part of the unparsed input. For example, in the rule

```
Variable:  identifier '(' ARBNO(expression,',')")'|identifier
```

(a variable may be either a subscripted or an unsubscripted identifier), assuming that 'identifier' has a normal definition and that a semantic routine is specified for it, 'identifier' will be invoked twice if the input string consists of an unsubscripted identifier. Both times it is invoked for the same parsed result (the identifier in the input), the first time as part of subscripted identifier.

After the first invocation of the semantic routine has returned, the first alternative will fail because a left parenthesis will not be found. The input pointer will be backed up past the identifier in the input stream, and the second alternative will be tried. The identifier will be reparsed, and the semantic routine reinvoked.

To avoid this problem, the PARSE_REQUEST can be given as:

```
Variable:  identifier('('ARBNO(expression,',')')'|)
```

(A variable is an identifier followed optionally by a subscript.) Here the identifier is parsed only once.

When using the minus sign (meaning no blanks may be between two PARSE_ATOMS) as the last element in the separator (second argument) of an ARBNO function, care must be used if the repetition string (first argument of ARBNO) has alternatives. The minus sign would apply to the first alternative in the repetition string since it always applies to only the next PARSE_ELEMENT. Normally, the minus sign is meant to apply to each alternative; this can be accomplished by enclosing the repetition string in angle brackets (thus making it a PARSE_GROUP and, hence, a single PARSE_ELEMENT).

VI. OPTIMIZATION

The user can do several things to speed up the parse and to reduce the amount of space it requires.

A) When defining a heavily used PARSE_REQUEST consisting of only one-character literals (e.g., the definition of 'letter'), use the routine COMPILE_PARSE_REQUEST rather than DEFINE_PARSE_REQUEST. This causes a translate table to be built and allows the alternatives to be tested in parallel simultaneously. This can greatly affect efficiency.

B) When specifying one alternative that is a prefix or a suffix of another, factor out the common portion and specify the rest as an option. For example,

```
A1 A2 A3|A1 A2
```

should be specified instead as

```
A1 A2(A3|)
```

This is especially important if A1, A2, or both are complex PARSE_REQUESTS as it can save extensive reparsing.

C) When possible, nested recursive (either left or right) definitions of parse-requests should be changed to iterative, or iterative recursive, definitions. For example, instead of defining number as:

```
Number:  Number digit|digit      (left recursive)
```

or as:

```
Number:  digit Number|digit      (right recursive),
```

it can be defined as:

Number: ARBNO(digit,) (iterative definition).

D) Finally, the ARB function should not be used more than is necessary since its use may involve large amounts of reparsing.


BLANK PAGE

Appendix A

EXAMPLE

```
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */
```

```
1               TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */
                   /* THIS PROGRAM PROVIDES AN ON-LINE SYNTAX CHECKER.
                   IT ALLOWS THE USER TO SPECIFY HIS SYNTAX IN APAREL FORMAT
                   AND TO TEST THIS SYNTAX AGAINST INPUT HE SUPPLIES ON-LINE.
                   THIS PROGRAM IS ITSELF WRITTEN IN APAREL AND USES APAREL
                   TO PARSE THE USERS COMMANDS. ITS SEMANTIC ROUTINES THEN
                   MAKE USE OF APAREL AGAIN TO DEFINE OR REDEFINE A RULE
                   SPECIFIED BY THE USER- OR TO TEST A RULE ON SOME INPUT
                   SUPPLIED BY THE USER.
                      THE ON-LINE INTERACTION IS SUPPLIED BY A SET OF PL/1
                   CALLABLE SUBROUTINES(SUPPLIED BY DICK WEXELBLATT OF BELL
                   LABS) WHICH INTERFACE WITH THE IBM 2260 ALPHANUMERIC DISPLAY
                   UNIT. */


                   /* THESE CALLS TO APAREL DEFINE THE SYNTAX LANGUAGE USED
                   BY THIS SYNTAX-TESTER. THE PARSE_REQUEST_NAMES ARE PURPOSELY
                   CHOSEN LONG SO THAT THE USER WILL NOT INADVERTANLY REDEFINE
                   THEM WHEN DEFINING HIS OWN LANGUAGE. */
2                  CALL DEFINE_PARSE_REQUEST(
                      'DEBUG_SYNTAX: <NEW.|> RULE. | INPUT.FOR.<RULE.!>
                      TEST_SYNTAX_NAME.IS.| TRACE | DISPLAY.<PARSE|>.RULES.
                      <FROM.TEST_SYNTAX_NAME|> | DISPLAY.TEST_SYNTAX_NAME
                      | LIST | PUNCH | READ | DELETE.TEST_SYNTAX_NAME | STOP | CLEAR
                   | FINISH',
                      NUL,DEBUG_SYNTAX_OPTION);
3                  CALL DEFINE_PARSE_REQUEST(
                      'TEST_SYNTAX_NAME: TEST_SYNTAX_LETTER-<ARBNO(<TEST_SYNTAX_LETTER
                      |TEST_SYNTAX_DIGIT>,-)!>',
                      SIMPLE_VARIABLE,SIMPLE_VARIABLE_OPTION);
4                  CALL COMPILE_PARSE_REQUEST(
                      'TEST_SYNTAX_LETTER:A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T
                      |U|V|W|X|Y|Z|_|#|@',
                      LETTER,LETTER_OPTION);
5                  CALL COMPILE_PARSE_REQUEST(
                      'TEST_SYNTAX_DIGIT: 0|1|2|3|4|5|6|7|8|9',
                      DIGIT,DIGIT_OPTION);
6                  CALL DEFINE_PARSE_REQUEST(
                      'PARSE_RULE_NAME:',NUL,PARSE_RULE_NAME_OPTION);
                   /* THESE PARSE_REQUEST DEFINITIONS ARE INCLUDED SO THAT
                   THE ON-LINE USER NEED NOT DEFINE THESE COMMONLY USED
                   PARSE_REQUESTS UNLESS HE WISHES TO MAKE A NON-STANDARD
                   DEFINITION. */
7                  DECLARE
                   (VARIABLE,SUBSCRIPT,SIMPLE_VARIABLE,ALPHANUMERIC,
                   BOOLEAN_EXPRESSION,RELATIONAL_OPERATOR,LOGICAL_OPERATOR,
                   NUMBER,LETTER,DIGIT,EXPRESSION,UNARY_OPERATOR,OPERATOR)
```

```
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

      CHARACTER(50) VARYING,
    (VARIABLE_OPTION,SUBSCRIPT_OPTION,SIMPLE_VARIABLE_OPTION,
    BOOLEAN_EXPRESSION_OPTION,RELATIONAL_OPERATOR_OPTION,
    LOGICAL_OPERATOR_OPTION,
    ALPHANUMERIC_OPTION,NUMBER_OPTION,LETTER_OPTION,
    EXPRESSION_OPTION,UNARY_OPERATOR_OPTION,OPERATOR_OPTION,
    DIGIT_OPTION) BINARY FIXED,
    NUL CHARACTER(0) EXTERNAL,
    SPECIFIED BINARY FIXED INITIAL(1),
    UNSPECIFIED BINARY FIXED INITIAL(0),
    SUCCESSFULLY BINARY FIXED INITIAL(1),
    UNSUCCESSFULLY BINARY FIXED INITIAL(0);

 8  CALL DEFINE_PARSE_REQUEST(
       'VARIABLE: SIMPLE_VARIABLE <
       ''('' ARBNO(<SUBSCRIPT: EXPRESSION>,'','','') '')'' | >',
       VARIABLE,VARIABLE_OPTION);
 9  CALL DEFINE_PARSE_REQUEST(
       'SUBSCRIPT:',SUBSCRIPT,SUBSCRIPT_OPTION);
10  CALL DEFINE_PARSE_REQUEST(
       'SIMPLE_VARIABLE: LETTER-< ARBNO(ALPHANUMERIC,-) | >',
       SIMPLE_VARIABLE,SIMPLE_VARIABLE_OPTION);
11  CALL DEFINE_PARSE_REQUEST(
       'ALPHANUMERIC: LETTER | DIGIT ',
       ALPHANUMERIC,ALPHANUMERIC_OPTION);
12  CALL DEFINE_PARSE_REQUEST(
       'NUMBER: ARBNO(DIGIT,-)',
       NUMBER,NUMBER_OPTION);
13  CALL COMPILE_PARSE_REQUEST(
       'LETTER: A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|
          X|Y|Z|_|#|@',LETTER,LETTER_OPTION);
14  CALL COMPILE_PARSE_REQUEST(
       'DIGIT: 0|1|2|3|4|5|6|7|8|9',
       DIGIT,DIGIT_OPTION);
15  CALL DEFINE_PARSE_REQUEST(
       'EXPRESSION: ARBNO(EXPRESSION,OPERATOR)
       | VARIABLE | NUMBER | ''('' EXPRESSION '')''
       |  UNARY_OPERATOR  EXPRESSION  ',EXPRESSION,
       EXPRESSION_OPTION);
16  CALL DEFINE_PARSE_REQUEST(
       'OPERATOR: + | ''-'' | * | ''/'' | ''||'' | **',
       OPERATOR,OPERATOR_OPTION);
17  CALL DEFINE_PARSE_REQUEST(
       'UNARY_OPERATOR: + | ''-'' | ''¬'' ',UNARY_OPERATOR,
       UNARY_OPERATOR_OPTION);
18  CALL DEFINE_PARSE_REQUEST(
```

```
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */


                  'BOOLEAN_EXPRESSION: ARBNO(BOOLEAN_EXPRESSION,LOGICAL_OPERATOR)
                  | ( BOOLEAN_EXPRESSION ) | EXPRESSION RELATIONAL_OPERATOR
                  EXPRESSION',
                  BOOLEAN_EXPRESSION,BOOLEAN_EXPRESSION_OPTION);
19                CALL DEFINE_PARSE_REQUEST(
                  'RELATIONAL_OPERATOR: ''<'' | ''>'' | = | <= | >= | ''¬<''
                     | ''¬>'' | ''¬=='' ',
                  RELATIONAL_OPERATOR,RELATIONAL_OPERATOR_OPTION);
20                CALL DEFINE_PARSE_REQUEST(
                  'LOGICAL_OPERATOR: ''|'' | & | ''¬'' ',
                  LOGICAL_OPERATOR,LOGICAL_OPERATOR_OPTION);


21                UNSPEC(START_SYMBOL)='01001010'B;
22             OPEN_SCOPE:
                  CALL GOPEN('SCOPE    ');
                     /* OPEN THE USER TERMINAL FOR INTERACTION */
23             INITIALIZE_SCREEN:
                  CALL GWRITE(EWL,UNIT,'APAREL ONLINE SYNTAX TESTER READY.');
                     /* ERASE THE SCREEN & WRITE A LINE(EWL) ON LINE 0(THE
                     FIRST CHARACTER OF THE MESSAGE) */
24             WAIT:
                  IF ¬GTEST(ANY_UNIT) THEN /* INTERRUPT IS NOT PENDING FROM USER */
25                   CALL GWAIT(ANY_UNIT); /* WAIT FOR ANY INTERRUPT */
26             READ_REQUEST:
                  CALL GREADV(SMI,UNIT,REQUEST); /* READ REQUEST */
27             ECHO_REQUEST:
                  MESSAGE=LINE(1)||' '||REQUEST;
28                CALL GWRITE(EWL,UNIT,MESSAGE);
                     /* REWRITE THE INPUT REQUEST ON LINE 1 OF DISPLAY */
29             DECODE_REQUEST:
                  SIMPLE_VARIABLE='';
30                CALL PARSE(REQUEST,'DEBUG_SYNTAX',REMAINING_INPUT);
                     /* PARSE THE INPUT(REQUEST) USING THE PARSE_REQUEST
                     'DEBUG_SYNTAX' AND PUT THE REMAINING INPUT IN STRING
                     REMAINING_INPUT */
31                   GO TO ROUTINE(DEBUG_SYNTAX_OPTION);
                     /* USE DEBUG_SYNTAX_OPTION, SET BY THE PARSE INVOKED ABOVE
                     AS AN N-WAY SWITCH TO GO TO THE PROPER SEMANTIC ROUTINE */


32             ROUTINE(0): /* ILLEGAL INPUT */
                  MESSAGE='ILLEGAL INPUT FOR THE APAREL ONLINE SYNTAX TESTER.';
33                GO TO PROCESSING_COMPLETE;


34             ROUTINE(1): /* NEW PARSE RULE */
```

```
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */


                  CALL PARSE(REMAINING_INPUT,'SIMPLE_VARIABLE',
                     REMAINING_INPUT2); /* GET PARSE NAME */
35                CALL FIND_NAME; /* SEARCH FOR PARSE NAME */
36                IF I=0 THEN DO;
                     /* NAME NOT FOUND. CREATE NEW PARSE RULE */
38                   DO I=1 TO PARSE_RULE_INDEX;
39                      IF PARSE_RULE(I)='' THEN /* EMPTY SPACE IN TABLE FOUND */
40                         GO TO DEFINE_RULE;
41                      END;
42                   PARSE_RULE_INDEX=PARSE_RULE_INDEX+1;
43                   I=PARSE_RULE_INDEX;
44             DEFINE_RULE:
                     MESSAGE='NEW PARSE RULE ACCEPTED.';
45                   END;
46                ELSE /* RULE ALREADY EXISTS, CHANGE IT */
46                   MESSAGE='PARSE RULE '||SIMPLE_VARIABLE||' HAS BEEN REDEFINED.';
47                PARSE_RULE(I)=REMAINING_INPUT;
48                PARSE_RESULT(I)='';
49                PARSE_RESULT_OPTION(I)=0;
50                CALL DEFINE_PARSE_REQUEST(
                     REMAINING_INPUT,PARSE_RESULT(I),
                     PARSE_RESULT_OPTION(I));
51                GO TO PROCESSING_COMPLETE;


52             ROUTINE(2): /* TEST INPUT FOR RULE */
                  CALL FIND_NAME; /* SEARCH FOR RULE NAME */
53                IF I>=0 THEN DO; /* RULE FOUND */
55                IF TRACE=1 THEN
56                   CALL TRACE_PARSE; /* TRACE THIS PARSE */
57                CALL PARSE(REMAINING_INPUT,SIMPLE_VARIABLE,REMAINING_INPUT2);
58                IF TRACE=1 THEN DO;
60                   CALL TRACE_PARSE; /* TURN TRACE BACK OFF */
61                   PUT PAGE;
62                   DO J=1 TO PARSE_RULE_INDEX;
63                      PUT SKIP(2) LIST('PARSE RULE=',PARSE_RULE(J));
64                      IF PARSE_RESULT_OPTION(J)=UNSPECIFIED THEN
65                         MESSAGE='PARSE RULE UNSUCCESSFUL';
66                      ELSE
66                         MESSAGE='ALTERNATIVE'||PARSE_RESULT_OPTION(J)
                              ||' SUCCESSFUL';
67                      PUT SKIP LIST(MESSAGE);
68                      PUT SKIP LIST('PARSE RESULTS='||PARSE_RESULT(J));
69                      END;
70                   PUT PAGE;
71                   END;
72                IF PARSE_RESULT_OPTION(I)=UNSPECIFIED THEN
```

```
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

73                   MESSAGE='PARSE OF RULE '||SIMPLE_VARIABLE||' UNSUCCESSFUL.';
74                ELSE DO; /* PARSE WAS SUCCESSFUL */
75                   MESSAGE=LINE(2)||'ALTERNATIVE'||PARSE_RESULT_OPTION(I)||
                        ' SUCCESSFUL.';
76                   CALL GWRITELNE,UNIT,MESSAGE);
77                   MESSAGE=LINE(3)||'PARSED INPUT='||PARSE_RESULT(I);
78                   CALL GWRITELNE,UNIT,MESSAGE);
79                   MESSAGE=LINE(4)||'REMAINING INPUT='||REMAINING_INPUT2;
80                   CALL GWRITELNE,UNIT,MESSAGE);
81                   MESSAGE='INPUT PARSED SUCCESSFULLY BY RULE '||
                        SIMPLE_VARIABLE;
82                   END;
83                   END;
84                ELSE /* PARSE RULE NOT FOUND */
             RULE_DOES_NOT_EXIST:
                   MESSAGE='PARSE RULE '||SIMPLE_VARIABLE||' DOES NOT EXIST.';
85                GO TO PROCESSING_COMPLETE;


86             ROUTINE(3): /* TRACE PARSE */
                  TRACE=1-TRACE; /* FLIP TRACE SWITCH  */
87                MESSAGE='TRACE OF PARSE SWITCH FLIPPED.';
88                GO TO PROCESSING_COMPLETE;


89             ROUTINE(4): /* DISPLAY LIST OF PARSE RULES */
90                IF SIMPLE_VARIABLE¬='' THEN DO;
                     /* START DISPLAY FROM NAMED RULE */
91                   CALL FIND_NAME; /* SEARCH FOR RULE */
92                   IF I¬=0 THEN
93                      PARSE_RULE_POSITION=I;
94                   ELSE
94                      GO TO RULE_DOES_NOT_EXIST;
95                   END;
96                ELSE /* RULE NOT SPECIFIED,CONTINUE FROM PRESENT POSITION */
96                   IF PARSE_RULE_POSITION>PARSE_RULE_INDEX THEN
97                      PARSE_RULE_POSITION=1; /* START OVER */
98                DO I=2 TO 10 WHILE(PARSE_RULE_POSITION<=PARSE_RULE_INDEX);
99                   CALL GWRITE(LNE,UNIT,LINE(I)||PARSE_RULE(
                        PARSE_RULE_POSITION));
100                  IF LENGTH(PARSE_RULE(PARSE_RULE_POSITION))>80 THEN
101                     I=I+1; /* SKIP AN EXTRA LINE */
102                  PARSE_RULE_POSITION=PARSE_RULE_POSITION+1;
103                  END;
104               MESSAGE='CONSECUTIVE PARSE RULES DISPLAYED.';
105               GO TO PROCESSING_COMPLETE;
```

```
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */


106             ROUTINE(5): /* DISPLAY INDIVIDUAL PARSE RULE */
                   CALL FIND_NAME; /* SEARCH FOR PARSE RULE */
107                IF I=0 THEN GO TO RULE_DOES_NOT_EXIST;
109                ELSE DO;
                         /* PARSE RULE FOUND */
110                      CALL GWRITE(LNE,UNIT,LINE(2)||'PARSE RULE='
                            ||PARSE_RULE(I));
111                      IF PARSE_RESULT_OPTION(I)=UNSPECIFIED THEN
112                         MESSAGE=LINE(4)||'PARSE RULE '||SIMPLE_VARIABLE
                               ||' UNSUCCESSFUL';
113                      ELSE /* PARSE WAS SUCCESSFUL */
113                         MESSAGE=LINE(4)||'ALTERNATIVE'||PARSE_RESULT_OPTION(I)
                               ||' OF PARSE RULE '||SIMPLE_VARIABLE||' SUCCESSFUL';
114                      CALL GWRITE(LNE,UNIT,MESSAGE);
115                      CALL GWRITE(LNE,UNIT,LINE(6)||'PARSE RESULTS='
                            ||PARSE_RESULT(I));
116                      MESSAGE='PARSE RULE '||SIMPLE_VARIABLE||' DISPLAYED.';
117                      GO TO PROCESSING_COMPLETE;
118                      END;


119             ROUTINE(6): /* LIST */
                   DO I=1 TO PARSE_RULE_INDEX;
120                   IF PARSE_RULE(I)¬='' THEN
121                      PUT SKIP(2) LIST(PARSE_RULE(I));
122                   END;
123                PUT PAGE;
124                MESSAGE='ALL PARSE RULES LISTED.';
125                GO TO PROCESSING_COMPLETE;


126             ROUTINE(7): /* PUNCH */
                   DO I=1 TO PARSE_RULE_INDEX;
127                   IF PARSE_RULE(I)¬='' THEN
128                      PUT FILE(PUNCH) SKIP(2) LIST(PARSE_RULE(I));
129                   END;
130                MESSAGE='ALL PARSE RULES PUNCHED.';
131                GO TO PROCESSING_COMPLETE;


132             ROUTINE(8): /* READ */
133                ON ENDFILE(SYSIN) GO TO PROCESSING_COMPLETE;
134                MESSAGE='PARSE RULES HAVE BEEN READ IN.';
135                DO WHILE(PARSE_RULE_INDEX<=HBOUND(PARSE_RULE,1));
136                   GET LIST(REMAINING_INPUT);
137                   CALL PARSE(REMAINING_INPUT,'SIMPLE_VARIABLE',
```

```
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */


                     REMAINING_INPUT2);
138                  CALL FIND_NAME; /* SEARCH FOR PARSE NAME */
139                  IF I=0 THEN DO;
                        /* RULE DOES NOT EXIST YET */
141                     DO I=1 TO PARSE_RULE_INDEX;
142                        IF PARSE_RULE(I)='' THEN GO TO DEFINE_RULE1;
144                        END;
145                     PARSE_RULE_INDEX=PARSE_RULE_INDEX+1;
146                     I=PARSE_RULE_INDEX;
147                     END;
148               DEFINE_RULE1:
                     PARSE_RULE(I)=REMAINING_INPUT;
149                  PARSE_RESULT(I)='';
150                  PARSE_RESULT_OPTION(I)=0;
151                  CALL DEFINE_PARSE_REQUEST(PARSE_RULE(I),
                        PARSE_RESULT(I),
                        PARSE_RESULT_OPTION(I));
152                  END;
153               MESSAGE='IMPLEMENTATION RESTRICTION: YOU HAVE TOO MANY RULES.';
154               GO TO PROCESSING_COMPLETE;


155               ROUTINE(9): /* DELETE */
                     CALL FIND_NAME;
156                  IF I=0 THEN GO TO RULE_DOES_NOT_EXIST;
158                  CALL DELETE_PARSE_REQUEST(SIMPLE_VARIABLE);
159                  PARSE_RULE(I)=''; /* INDICATE RULE DELETED */
160                  MESSAGE='PARSE RULE '||SIMPLE_VARIABLE||' HAS BEEN DELETED.';
161                  GO TO PROCESSING_COMPLETE;


162               ROUTINE(10): /* STOP */
                     CALL GCLOSE;
163                  UNSPEC(EOM)='01101010'B;
164                  CALL SGSMESS(128+64+32,' STOP'||EOM); /* ISSUE STOP
                        MESSAGE TO SGS, SET CONTROL BYTE FOR INTERPRET MESSAGE
                        AS COMMING FROM USER'S SCOPE, INPUT AND OUTPUT FROM THERE
                        ALSO */
                        GO TO OPEN_SCOPE;


166               ROUTINE(11): /* CLEAR */
                     DO I=1 TO PARSE_RULE_INDEX;
167                     CALL PARSE(PARSE_RULE(I),'SIMPLE_VARIABLE',REMAINING_INPUT);
168                     CALL DELETE_PARSE_REQUEST(SIMPLE_VARIABLE);
169                     PARSE_RULE(I)=''; /* INDICATE PARSE RULE DELETED */
170                     END;
```

```
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

171                  PARSE_RULE_INDEX=0;
172                  MESSAGE='ALL PARSE RULES HAVE BEEN DELETED.';
173                  GO TO PROCESSING_COMPLETE;


174               ROUTINE(12): /* FINISH */
                     RETURN;


175               PROCESSING_COMPLETE:
                     MESSAGE=LINE(0)||MESSAGE;
176                  CALL GWRITE(LINE,UNIT,MESSAGE||START_SYMBOL);
177                  GO TO WAIT;


178               FIND_NAME: PROCEDURE;
179                  REMAINING_INPUT2='PARSE_RULE_NAME:'''||SIMPLE_VARIABLE||''':';
                        /* CREATE PARSE RULE TO FIND NAME */
180                  DO I=1 TO PARSE_RULE_INDEX;
181                     CALL PARSE(PARSE_RULE(I),REMAINING_INPUT2,REMAINING_INPUT3);
182                     IF PARSE_RULE_NAME_OPTION=SPECIFIED THEN RETURN;
184                     END;
185                  I=0; /* PARSE RULE NOT FOUND */
186                  END FIND_NAME;


187               DECLARE
                     PUNCH FILE PRINT,
                     REMAINING_INPUT2 CHARACTER(80) VARYING,
                     REMAINING_INPUT3 CHARACTER(80) VARYING,
                     ROUTINE(0:15) LABEL,
                     PARSE_RULE_POSITION BINARY FIXED INITIAL(1),
                     SGSMESS ENTRY(BINARY FIXED,),
                     EOM CHARACTER(1),
                     PARSE_RULE(50) CHARACTER(160) VARYING,
                     PARSE_RULE_INDEX BINARY FIXED INITIAL(0),
                     DEBUG_SYNTAX_OPTION BINARY FIXED,
                     PARSE_RULE_NAME_OPTION BINARY FIXED,
                     PARSE_RESULT(50) CHARACTER(100) VARYING,
                     PARSE_RESULT_OPTION(50) BINARY FIXED,
                     TRACE BINARY FIXED INITIAL(0),
                     J BINARY FIXED,
                     I BINARY FIXED;
188               DECLARE
                        GTEST RETURNS(BIT(1)),
                        UNIT BINARY FIXED INITIAL(1),
                        SMI BINARY FIXED INITIAL(1),
```

```
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

                RSB BINARY FIXED INITIAL(2),
                LNE BINARY FIXED INITIAL(2),
                EWL BINARY FIXED INITIAL(3),
                ECHO_TYPE BINARY FIXED INITIAL(3),
                ANY_UNIT BINARY FIXED INITIAL(255),
                CLEAR BINARY FIXED INITIAL(256),
                ONSCOPE BINARY FIXED INITIAL(0) EXTERNAL,
                GCODE BINARY FIXED INITIAL(0) EXTERNAL,
                GCOUNT BINARY FIXED INITIAL(0) EXTERNAL,
                LINE(0:11) CHARACTER(1) INITIAL('0','1','2',
                      '3','4','5','6','7','8','9','A','B'),
                MESSAGE CHARACTER(81),
                INITIAL_MESSAGE CHARACTER(81),
                REQUEST CHARACTER(160) VARYING,
                REMAINING_INPUT CHARACTER(160) VARYING,
                START_SYMBOL CHARACTER(1),
                GWAIT ENTRY(BINARY FIXED);
189           END TEST_SYNTAX;
```

TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

ATTRIBUTE AND CROSS-REFERENCE TABLE

DCL NO.	IDENTIFIER	ATTRIBUTES AND REFERENCES
7	ALPHANUMERIC	AUTOMATIC,UNALIGNED,STRING(50),CHARACTER,VARYING 11
7	ALPHANUMERIC_OPTION	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 11
188	ANY_UNIT	AUTOMATIC,ALIGNED,INITIAL,BINARY,FIXED(15,0) 24,25
7	BOOLEAN_EXPRESSION	AUTOMATIC,UNALIGNED,STRING(50),CHARACTER,VARYING 18
7	BOOLEAN_EXPRESSION_OPTION	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 18
188	CLEAR	AUTOMATIC,ALIGNED,INITIAL,BINARY,FIXED(15,0)
	COMPILE_PARSE_REQUEST	EXTERNAL,ENTRY,DECIMAL,FLOAT(SINGLE) 4,5,13,14
187	DEBUG_SYNTAX_OPTION	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 2,31
29	DECODE_REQUEST	STATEMENT LABEL CONSTANT
	DEFINE_PARSE_REQUEST	EXTERNAL,ENTRY,DECIMAL,FLOAT(SINGLE) 2,3,6,8,9,10,11,12,15,16,17,18,19,20,50,151
44	DEFINE_RULE	STATEMENT LABEL CONSTANT 40
148	DEFINE_RULE1	STATEMENT LABEL CONSTANT 143
	DELETE_PARSE_REQUEST	EXTERNAL,ENTRY,DECIMAL,FLOAT(SINGLE) 158,168
7	DIGIT	AUTOMATIC,UNALIGNED,STRING(50),CHARACTER,VARYING 5,14
7	DIGIT_OPTION	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0)

TEST_SYNTAX: PROCEDURE OPTIONSIMAIN); /* FILE TESTSYN */

DCL NO.	IDENTIFIER	ATTRIBUTES AND REFERENCES
		5,14
27	ECHO_REQUEST	STATEMENT LABEL CONSTANT
188	ECHO_TYPE	AUTOMATIC,ALIGNED,INITIAL,BINARY,FIXED(15,0)
187	EOM	AUTOMATIC,UNALIGNED,STRING(1),CHARACTER 163,164
188	EWL	AUTOMATIC,ALIGNED,INITIAL,BINARY,FIXED(15,0) 23,28
7	EXPRESSION	AUTOMATIC,UNALIGNED,STRING(50),CHARACTER,VARYING 15
7	EXPRESSION_OPTION	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 15
178	FIND_NAME	ENTRY,DECIMAL,FLOAT(SINGLE) 35,52,91,106,138,155
	GCLOSE	EXTERNAL,ENTRY,DECIMAL,FLOAT(SINGLE) 162
188	GCODE	STATIC,EXTERNAL,ALIGNED,INITIAL,BINARY,FIXED(15,0)
188	GCOUNT	STATIC,EXTERNAL,ALIGNED,INITIAL,BINARY,FIXED(15,0)
	GOPEN	EXTERNAL,ENTRY,DECIMAL,FLOAT(SINGLE) 22
	GREADV	EXTERNAL,ENTRY,DECIMAL,FLOAT(SINGLE) 26
188	GTEST	EXTERNAL,ENTRY,STRING(1),BIT 24
188	GWAIT	EXTERNAL,ENTRY,DECIMAL,FLOAT(SINGLE) 25
	GWRITE	EXTERNAL,ENTRY,DECIMAL,FLOAT(SINGLE) 23,28,76,78,80,99,110,114,115,176
	HBOUND	GENERIC,BUILT-IN FUNCTION

TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

DCL NO.	IDENTIFIER	ATTRIBUTES AND REFERENCES
		135
187	I	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 36,38,39,43,47,48,49,50,52,53,72,75,77,92,93,98,99,101,151,167,170 111,113,115,119,120,121,126,127,128,139,141,142,146,148,149,152,151 151,151,156,159,166,167,169,180,181,185
188	INITIAL_MESSAGE	AUTOMATIC,UNALIGNED,STRING(81),CHARACTER
23	INITIALIZE_SCREEN	STATEMENT LABEL CONSTANT
187	J	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 62,63,64,66,68
	LENGTH	GENERIC,BUILT-IN FUNCTION 100
7	LETTER	AUTOMATIC,UNALIGNED,STRING(50),CHARACTER,VARYING 4,13
7	LETTER_OPTION	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 4,13
188	LINE	10:11)AUTOMATIC,UNALIGNED,INITIAL,STRING(1),CHARACTER 27,75,77,79,99,110,112,113,115,175
188	LNE	AUTOMATIC,ALIGNED,INITIAL,BINARY,FIXED(15,0) 76,78,80,99,110,114,115,176
7	LOGICAL_OPERATOR	AUTOMATIC,UNALIGNED,STRING(50),CHARACTER,VARYING 20
7	LOGICAL_OPERATOR_OPTION	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 20
188	MESSAGE	AUTOMATIC,UNALIGNED,STRING(81),CHARACTER 27,28,32,44,46,65,66,67,73,75,76,77,78,79,80,81,84,87,104,112,113 114,116,124,130,134,153,160,172,175,175,176
7	NUL	STATIC,EXTERNAL,UNALIGNED,STRING(0),CHARACTER 2,6
7	NUMBER	AUTOMATIC,UNALIGNED,STRING(50),CHARACTER,VARYING 12

```
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */
```

DCL NO.	IDENTIFIER	ATTRIBUTES AND REFERENCES
7	NUMBER_OPTION	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 12
188	ONSCOPE	STATIC,EXTERNAL,ALIGNED,INITIAL,BINARY,FIXED(15,0)
22	OPEN_SCOPE	STATEMENT LABEL CONSTANT 165
7	OPERATOR	AUTOMATIC,UNALIGNED,STRING(50),CHARACTER,VARYING 16
7	OPERATOR_OPTION	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 16
	PARSE	EXTERNAL,ENTRY,DECIMAL,FLOAT(SINGLE) 30,34,57,137,167,181
187	PARSE_RESULT	(50)AUTOMATIC,UNALIGNED,STRING(100),CHARACTER,VARYING 48,50,68,77,115,149,151
187	PARSE_RESULT_OPTION	(50)AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 49,50,64,66,72,75,111,113,150,151
187	PARSE_RULE	(50)AUTOMATIC,UNALIGNED,STRING(160),CHARACTER,VARYING 39,47,63,99,100,110,120,121,127,128,135,142,148,151,159,167,169,181
187	PARSE_RULE_INDEX	AUTOMATIC,ALIGNED,INITIAL,BINARY,FIXED(15,0) 38,42,42,43,62,96,98,119,126,135,141,145,145,146,166,171,180
187	PARSE_RULE_NAME_OPTION	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 6,182
187	PARSE_RULE_POSITION	AUTOMATIC,ALIGNED,INITIAL,BINARY,FIXED(15,0) 93,96,97,98,99,100,102,102
175	PROCESSING_COMPLETE	STATEMENT LABEL CONSTANT 33,51,85,88,105,117,125,131,133,154,161,173
187	PUNCH	FILE,EXTERNAL,PRINT 128
26	REAC_REQUEST	STATEMENT LABEL CONSTANT

TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

DCL NO.	IDENTIFIER	ATTRIBUTES AND REFERENCES
7	RELATIONAL_OPERATOR	AUTOMATIC,UNALIGNED,STRING(50),CHARACTER,VARYING 19
7	RELATIONAL_OPERATOR_OPTION	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 19
188	REMAINING_INPUT	AUTOMATIC,UNALIGNED,STRING(160),CHARACTER,VARYING 30,34,47,50,57,136,137,148,167
187	REMAINING_INPUT2	AUTOMATIC,UNALIGNED,STRING(80),CHARACTER,VARYING 34,57,79,137,179,181
187	REMAINING_INPUT3	AUTOMATIC,UNALIGNED,STRING(80),CHARACTER,VARYING 181
188	REQUEST	AUTOMATIC,UNALIGNED,STRING(160),CHARACTER,VARYING 26,27,30
187	ROUTINE	(0:15)AUTOMATIC,INITIAL,LABEL 32,34,52,86,89,106,119,126,132,155,162,166,174,31
188	RSB	AUTOMATIC,ALIGNED,INITIAL,BINARY,FIXED(15,0)
84	RULE_DOES_NOT_EXIST	STATEMENT LABEL CONSTANT 94,108,157
187	SGSMESS	EXTERNAL,ENTRY,DECIMAL,FLOAT(SINGLE) 164
7	SIMPLE_VARIABLE	AUTOMATIC,UNALIGNED,STRING(50),CHARACTER,VARYING 3,10,29,46,57,73,81,84,89,112,113,116,158,160,168,179
7	SIMPLE_VARIABLE_OPTION	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 3,10
188	SMI	AUTOMATIC,ALIGNED,INITIAL,BINARY,FIXED(15,0) 26
7	SPECIFIED	AUTOMATIC,ALIGNED,INITIAL,BINARY,FIXED(15,0) 182
188	START_SYMBOL	AUTOMATIC,UNALIGNED,STRING(1),CHARACTER 21,176

TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

DCL NO.	IDENTIFIER	ATTRIBUTES AND REFERENCES
7	SUBSCRIPT	AUTOMATIC,UNALIGNED,STRING(50),CHARACTER,VARYING 9
7	SUBSCRIPT_OPTION	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 9
7	SUCCESSFULLY	AUTOMATIC,ALIGNED,INITIAL,BINARY,FIXED(15,0)
	SYSIN	FILE,EXTERNAL 132,136
	SYSPRINT	FILE,EXTERNAL 61,63,67,68,70,121,123
1	TEST_SYNTAX	ENTRY,DECIMAL,FLOAT(SINGLE)
187	TRACE	AUTOMATIC,ALIGNED,INITIAL,BINARY,FIXED(15,0) 55,58,86,86
	TRACE_PARSE	EXTERNAL,ENTRY,DECIMAL,FLOAT(SINGLE) 56,60
7	UNARY_OPERATOR	AUTOMATIC,UNALIGNED,STRING(50),CHARACTER,VARYING 17
7	UNARY_OPERATOR_OPTION	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 17
188	UNIT	AUTOMATIC,ALIGNED,INITIAL,BINARY,FIXED(15,0) 23,26,28,76,78,80,99,110,114,115,176
	UNSPEC	GENERIC,BUILT-IN FUNCTION 21,163
7	UNSPECIFIED	AUTOMATIC,ALIGNED,INITIAL,BINARY,FIXED(15,0) 64,72,111
7	UNSUCCESSFULLY	AUTOMATIC,ALIGNED,INITIAL,BINARY,FIXED(15,0)
7	VARIABLE	AUTOMATIC,UNALIGNED,STRING(50),CHARACTER,VARYING 8
7	VARIABLE_OPTION	AUTOMATIC,ALIGNED,BINARY,FIXED(15,0) 8

TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

DCL NO.	IDENTIFIER	ATTRIBUTES AND REFERENCES
24	WAIT	STATEMENT LABEL CONSTANT 177

TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

AGGREGATE LENGTH TABLE

STATEMENT NO.	IDENTIFIER	LENGTH IN BYTES
188	LINE	12
187	PARSE_RESULT	5000
187	PARSE_RESULT_OPTION	200
187	PARSE_RULE	8000
187	ROUTINE	128

```
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */
```

STORAGE REQUIREMENTS.

THE STORAGE AREA FOR THE PROCEDURE LABELLED TEST_SYNTAX IS 16828 BYTES LONG.
THE STORAGE AREA FOR THE ON UNIT AT STATEMENT NO. 132 IS 144 BYTES LONG.
THE STORAGE AREA (IN STATIC) FOR THE PROCEDURE LABELLED FIND_NAME IS 352 BYTES LONG.
THE PROGRAM CSECT IS NAMED TESTTAX AND IS 9010 BYTES LONG.
THE STATIC CSECT IS NAMED TESTTAXA AND IS 4264 BYTES LONG.

```
        TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

   COMPILER DIAGNOSTICS.


ERRORS.

   IEM2867I          IMPLEMENTATION RESTRICTION. EXTERNAL NAME TEST_SYNTAX HAS BEEN TRUNCATED TO 7 CHARACTERS.

   IEM2867I          IMPLEMENTATION RESTRICTION. EXTERNAL NAME DEFINE_PARSE_REQUEST HAS BEEN TRUNCATED TO 7
                     CHARACTERS.

   IEM2867I          IMPLEMENTATION RESTRICTION. EXTERNAL NAME COMPILE_PARSE_REQUEST HAS BEEN TRUNCATED TO 7
                     CHARACTERS.

   IEM2867I          IMPLEMENTATION RESTRICTION. EXTERNAL NAME TRACE_PARSE HAS BEEN TRUNCATED TO 7 CHARACTERS.

   IEM2867I          IMPLEMENTATION RESTRICTION. EXTERNAL NAME DELETE_PARSE_REQUEST HAS BEEN TRUNCATED TO 7
                     CHARACTERS.


WARNINGS.

   IEM0227I          NO FILE/STRING OPTION SPECIFIED IN ONE OR MORE GET/PUT STATEMENTS. SYSIN/SYSPRINT HAS BEEN
                     ASSUMED IN EACH CASE.


END OF DIAGNOSTICS.
```

```
TEST_SYNTAX: PROCEDURE OPTIONS(MAIN); /* FILE TESTSYN */

COMPILE TIME          .72 MINS
```

Appendix B

BNF DEFINITION OF APAREL'S SYNTAX LANGUAGE

```
⟨PARSE_REQUEST⟩ := ⟨PARSE_DELIMINATOR⟩⟨PARSE_NAME⟩:
          ⟨PARSE_ALTERNATIVE_LIST⟩⟨PARSE_DELIMINATOR⟩
⟨PARSE_ALTERNATIVE_LIST⟩ := ⟨PARSE_ALTERNATIVE_NAME⟩
          ⟨PARSE_ELEMENT_LIST⟩ | ⟨PARSE_ALTERNATIVE_NAME⟩
          ⟨PARSE_ELEMENT_LIST⟩ '|' ⟨PARSE_ALTERNATIVE_LIST⟩
⟨PARSE_ELEMENT_LIST⟩ := ⟨PARSE_ELEMENT⟩ |
          ⟨PARSE_ELEMENT⟩;⟨PARSE_TIME_ROUTINE_NAME⟩ |
          ⟨PARSE_ELEMENT⟩⟨PARSE_ELEMENT_LIST⟩ |
          ⟨PARSE_ELEMENT⟩.⟨PARSE_ELEMENT_LIST⟩
⟨PARSE_ELEMENT⟩ := ⟨PARSE_ATOM⟩ | ⟨PARSE_GROUP⟩
⟨PARSE_GROUP⟩ := '⟨' ⟨PARSE-ALTERNATIVE_LIST⟩ '⟩' |
          '⟨' ⟨PARSE_NAME⟩:⟨PARSE_ALTERNATIVE_LIST⟩ '⟩'
⟨PARSE_ATOM⟩ := ⟨PARSE_NAME⟩ | ⟨TEXT_LITERAL⟩
⟨PARSE_NAME⟩ := ⟨PL/1 IDENTIFIER⟩
⟨PARSE_ALTERNATIVE_NAME⟩ := (⟨PL/1 IDENTIFIER⟩)
⟨PARSE_DELIMINATOR⟩ := ::
⟨PARSE_TIME_ROUTINE_NAME⟩ := ⟨NAME OF A PL/1 BIT VALUED FUNCTION ⟩
```

REFERENCES

1. Balzer, R. M., and D. J. Farber, *APAREL--A Parse-Request Language*, The RAND Corporation, RM-5611-1-ARPA, September 1969.

2. *IBM System/360, Principles of Operation*, Form A22-6821, IBM Corporation.